Noise-contrastive estimation of normalising constants and GANs

Fonctions génériques

Algorithme d'Hasting

Utilité : simuler selon $p_m(.,\psi)$ pour un paramètre ψ choisi.

Argument	Type	Exemple	Indication
x	vecteur	rcauchy(100, 0, 1)	notre échantillon de densité inconnue
n	entier	100	taille de la simulation
psi	vecteur	c(0,1)	paramètres de la fonction h
h	fonction		fonction qui retourne $\overline{p_m}(.,\psi)$

```
hasting = function(x, n, psi, h){
  y = c()
  y = append(y, sample(sample(x, 1)))
  for (i in 2:n){
    y_ = rnorm(1, y[i-1], 1)
    u = runif(1)
    if ( u <=
         (h(y_,psi) * dnorm(y_, y[i-1], 1))
         /(h(y[i-1],psi) * dnorm(y[i-1], y_, 1))
    ){
      y = append(y, y_)}
    else {
      y = append(y, y[i-1])
    }
  }
  return (y)
}
```

Note : on peut très certainement écrire sous forme matricielle cette fonction pour une meilleure performance.

MC MLE

Utilité : retourne une estimation des paramètres selon la méthode décrite dans le papier de Geyer.

```
mc_mle = function(x, n, psi, h){

  y = hasting(x, n, psi, h)

  L = function(theta){
    return(sum(log(h(x,theta)/h(x,psi))) - n*log(mean(h(y,theta)/h(y,psi))))
  }

  theta = optim(
    par = rep(1,length(psi)),
    gr = "CG",
```

```
    control = list(fnscale=-1),
    fn = L
  )$par

  return(theta)
}
```

NCE

Utilité : Retourne l'estimation de la constante et des paramètres.

Argument	Type	Exemple	Indication
x	vecteur	rcauchy(100, 0, 1)	notre échantillon de densité inconnue
law_y	fonction	rnorm	fonction qui retourne un échantillon suivant la loi p_n
n	entier	100	taille de l'échantillon de bruit suivant la loi p_n
params_y	vecteur	c(0,1)	arguments de la fonction law_y
log_pm	fonction		fonction qui retourne le logarithme de la densité p_m
log_pn	fonction		fonction qui retourne le logarithme de la densité p_n
nb_of_params	entier	3	taille de θ, vaut habituellement 2 ou 3
method	string	"CG"	méthode d'optimisation, habituellement "CG" ou "BFGS"

```
nce = function(x, law_y, n, params_y, log_pm, log_pn, nb_of_params, methode = "CG"){

  y = do.call(law_y,c(list(n),params_y))

  m = length(x)

  h = function(u, theta){
    return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
  }

  J = function(theta){
    return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
  }

  theta = optim(
    par = initialisation,
    gr = methode,
    control = list(fnscale=-1),
    fn = J
  )$par

  return(c(exp(-theta[1]), theta[-1]))
}
```